Scala & JEE

Ant Kutschera, ITIC 2012
www.maxant.ch

Quick Survey?

- Who uses Scala?
- Who wants to use Scala?
- Who thinks it's a waste of time?
- Who thinks it isn't time yet?

Why Scala?

- Functional Programming
  - Support for monads, pattern matching, immutability built into language
- 3M€ investment into Typesafe
  - Someone is betting on it
- Concise language
  - Less code, more maintainable?
- More features
  - Actors, superior collections, parallel collections, DSL support, XML support
- Fun!

Scala Environment

- Everything is new
  - Build tool => sbt
  - Test framework => scalatest, specs2
  - Database Frameworks => squeryl, ScalaQuery / SLICK
  - Web => Lift / Play!
- I can't leverage my JEE knowledge
- Maturity of new products?

Java Enterprise Edition

- JSF 2 => Primefaces, etc. have full Ajax Support incl. Server Push
- EJB => Business Tier
- Web Services (SOAP & REST)
- JPA
- JCA for Integration

- Cross cutting: Security, transactions, resources, concurrency, scalability, reliability, configuration, etc.

So let's write an EJB

```
package ch.maxant.scalabook.services
import javax.ejb.Stateless

@Stateless
class DoesNotWorkService {

    @Inject var log: Logger = null
    @PersistenceContext(unitName="theDatabase") var em: EntityManager = null
    @Resource var ctx: SessionContext = null
    @EJB var userService: UserService = null

    def foo(bar: String) = {
        //something useful
    }

}
```

So let's write an EJB

```
package ch.maxant.scalabook.services
import javax.ejb.Stateless

@Stateless
class DoesNotWorkService {

    @Inject var log: Logger = null
    @PersistenceContext(unitName="theDatabase") var em: EntityManager = null
    @Resource var ctx: SessionContext = null
    @EJB var userService: UserService = null

    def foo(bar: String) = {
        //something useful
    }

}
```

- Doesn't work!

```
org.jboss.as.server.deployment.DeploymentUnitProcessingExcept
ion: JBAS014544:
No EJB found with interface of type
'ch.maxant.scalabook.services.DoesNotWorkService' for binding
ch.maxant.scalabook.services.DoesNotWorkClient/svc
```

So let's write an EJB

```
@Stateless
@ScalaSignature(bytes="")
public class DoesNotWorkService
  implements ScalaObject
{

  @Inject
  private Logger log;

  @PersistenceContext(unitName="theDatabase")
  private EntityManager em;

  @Resource
  private SessionContext ctx;

  @EJB
  private UserService userService;
```

Generated decompiled bytecode

Required modification

```
package ch.maxant.scalabook.services
import javax.ejb.Stateless

@Stateless
@LocalBean
class DoesNotWorkService {

    @Inject var log: Logger = null
    @PersistenceContext(unitName="theDatabase") var em: EntityManager = null
    @Resource var ctx: SessionContext = null
    @EJB var userService: UserService = null

    def foo(bar: String) = {
        //something useful
    }

}
```

Business code example

```
def reserveOffer(event: EventOffer, tarifs: Seq[Tarif]) = {

    var reservations = tarifs.par.map{ t =>
        getAdapter(t.bookingSystem) match {
            case Some(a) => a.reserveOffer(event, t)
            case _ => throw new
                    IllegalStateException(
                            "Unknown adapter " +
                            t.bookingSystem)
        }
    }

    //so that the caller isnt surprised, we
    //should convert this back to a non-parallel sequence
    reservations.seq
}
```

- Uses: parallel collections, pattern matching, monadic function

More code examples

```
def distinctCities = {
    getTeasers().map(_.address.city).distinct.sorted
}
```

Collections API (monads)

DSL

```
val premium = forEveryTicketIn(shoppingCart){ ticket =>

    if( ticket.value > 50.CHF)

        14 .percent of ticket.value

    else

        11 .percent of ticket.value

} subtract (3.10 .if_total > 10.CHF )
```

More code examples

```
case class WeatherData(
        lat: Double,
        long: Double,
        city: String,
        maxTemperaturePerMonth: List[Double]
){}
```

Simple model class

MapReduce, in parallel

```
val cities = List(
        WeatherData(0.0, 0.0, "Heraklion", List(20.0, 21.1, 23.2, 24.1, 25.6, 27.
        WeatherData(0.0, 0.0, "Lausanne", List(4.3, 9.9, 16.5, 22.1, 25.6, 24.6,
)

//calculate world average maximum temperature in parallel
var avgMaxPerCity = cities.par.map(_.maxTemperaturePerMonth.reduce(_+_) / 12)
var avgMaxWorld = avgMaxPerCity.reduce(_+_) / avgMaxPerCity.size

println("avg is: " + avgMaxWorld)
```

Let's write some JSF

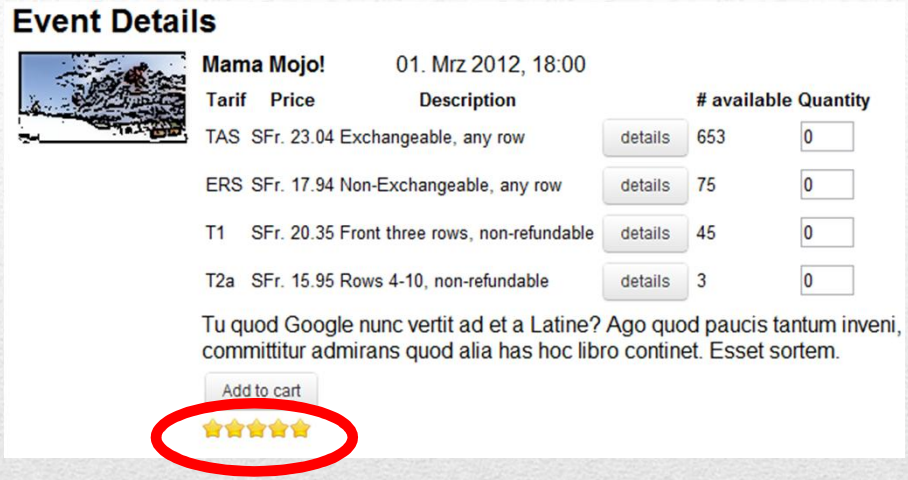
```
@Named
@RequestScoped
class Details {

    @Inject var log: Logger = null
    @Inject var model: Model = null
    @Inject var ctx: FacesContext = null
    @EJB var eventService: EventService = null
    @Inject var login: Login = null

    @BeanProperty var uid: String = null
    @BeanProperty var rating = -1
    @BeanProperty var ratingReadonly = false

    private var generalError: String = null

    @PostConstruct
    def init(): Unit = {
```




```
<td colspan="2">
    <p:rating
            id="rater"
            value="#{details.rating}"
            readonly="#{details.ratingReadonly}"
            cancel="false"
            >
        <p:ajax event="rate" listener="#{details.doRating}" update="rater" />
        <p:ajax event="cancel" listener="#{details.doRating}" update="rater" />
    </p:rating>
</td>
```

Let's write a JPA entity

```
@Entity
@Access(AccessType.FIELD)
case class Rule(

    @Id
    @TableGenerator(name="RULE_GENERATOR", table="RULE_SEQUENCE",  pkColumnValue="RULE")
    @GeneratedValue(strategy=GenerationType.TABLE, generator="RULE_GENERATOR")
    id: Long,

    name: String,
    expression: String,
    outcome: String,
    priority: Int,
    namespace: String,
    description: String
) {

    /** default no arg constructors are required by jpa */
    def this() = {
        this(0L, null, null, null, 0, null, null)
    }

    /** converts this rule into a rule engine rule */
    def convert() = new ch.maxant.rules.Rule(name, expression, outcome, priority, namespace, description)

}
```

Immutable, free #hashCode(), #equals(Object), #toString(), pattern matching, optimised copy constructor

Sidebar:

- Why is immutability interesting for persistence?
  - JDBC – required you to map from `ResultSet` to a class
  - Hibernate – no mapping… but change an attached object, and it results in an update SQL
    - Rule: Persistence layer objects may not be passed up to the business layer because we don't trust each other
      - Result: another mapper!
- JPA has the "merge" method, to merge a modified copy of an immutable object into the DB. Scala supports lean copy constructors.

Let's write a JPA entity

```
@Entity
@Access(AccessType.FIELD)
case class Rule(

    @Id
    @TableGenerator(name="RULE_GENERATOR", table="RULE_SEQUENCE",  pkColumnValue="RULE")
    @GeneratedValue(strategy=GenerationType.TABLE, generator="RULE_GENERATOR")
    id: Long,

    name: String,
    expression: String,
    outcome: String,
    priority: Int,
    namespace: String,
    description: String
) {

    /** default no arg constructors are required by jpa */
    def this() = {
        this(0L, null, null, null, 0, null, null)
    }

    /** converts this rule into a rule engine rule */
    def convert() = new ch.maxant.rules.Rule(name, expression, outcome, priority, namespace, description)

}
```

Immutable, free #hashCode(), #equals(Object), #toString(), pattern matching, optimised copy constructor

Let's write a JPA entity

```
@Entity
@Access(AccessType.FIELD)
@ScalaSignature(bytes="")
public class Rule
  implements Product, Serializable
{
  private final long id;
  private final String name;
  private final String expression;
  private final String outcome;
  private final int priority;
  private final String namespace;
  private final String description;
```

Generated decompiled bytecode

Where are the annotations?

A "case" class has no
bean-style accessor methods

Let's write a JPA entity

```
@Entity
@Access(AccessType.FIELD)
case class Rule(

    //dont forget the prefix!
    @JPAHelp.Id
    @JPAHelp.TableGenerator(name="RULE_GENER
    @JPAHelp.GeneratedValue(strategy=Generat
    id: Long,

    name: String,
    expression: String,
    outcome: String,
    priority: Int,
    namespace: String,
    description: String
) {
```

```
object JPAHelp {
    type Id = javax.persistence.Id @field
    type TableGenerator = javax.persistence.TableGenerator @field
    type GeneratedValue = javax.persistence.GeneratedValue @field
    type Column = javax.persistence.Column @field
    type ManyToOne = javax.persistence.ManyToOne @field
    type OneToMany = javax.persistence.OneToMany @field
    type ManyToMany = javax.persistence.ManyToMany @field
    type OrderBy = javax.persistence.OrderBy @field
    type MapKeyJoinColumn = javax.persistence.MapKeyJoinColumn @field
    type OrderColumn = javax.persistence.OrderColumn @field
    type Cacheable = javax.persistence.Cacheable @field
    type JoinColumn = javax.persistence.JoinColumn @field
    type Version = javax.persistence.Version @field
}
```

More complex example

```
    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    def updateRating(eventUid: String, rating: Int, userId: Long)= {
       //TODO replace with ScalaQuery!!
        val q1 = em.createQuery("""
UPDATE Rating r
SET
  r.numRatings = (SELECT r2.numRatings + 1 FROM Rating r2 WHERE r2.eventUid = :eventUid)
, r.sumRatings = (SELECT r3.sumRatings + :newRating FROM Rating r3 WHERE r3.eventUid = :eventUid)
WHERE
  r.eventUid = :eventUid
        """)
        q1.setParameter("eventUid", eventUid)
        q1.setParameter("newRating", rating.toLong)
        q1.executeUpdate

        //now block the user from doing another rating
        em.persist(RatingUserMapping(-1, eventUid, userId, rating))

        getRating(eventUid)
    }
```

More complex example

```
    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    def updateRating(eventUid: String, rating: Int, userId: Long)= {
       //TODO replace with ScalaQuery!!
        val q1 = em.createQuery("""
UPDATE Rating r
SET
  r.numRatings = (SELECT r2.numRatings + 1 FROM Rating r2 WHERE r2.eventUid = :eventUid)
, r.sumRatings = (SELECT r3.sumRatings + :newRating FROM Rating r3 WHERE r3.eventUid = :eventUid)
WHERE
  r.eventUid = :eventUid
        """)
        q1.setParameter("eventUid", eventUid)
        q1.setParameter("newRating", rating.toLong)
        q1.executeUpdate

        //now block the user from doing another rating
        em.persist(RatingUserMapping(-1, eventUid, userId, rating))

        getRating(eventUid)
    }
```

- Why use ORM?
- Type safety?
  - JPA isn't really type safe, even with the Criteria API

More complex example

- Impedance mismatch
  - Why solve SQL with OO?
- Java has libraries like JOOQ

```
  SELECT FIRST_NAME, LAST_NAME, COUNT(*)
    FROM AUTHOR
    JOIN BOOK ON AUTHOR.ID = BOOK.AUTHOR_ID
   WHERE LANGUAGE = 'DE'
     AND PUBLISHED > '2008-01-01'
GROUP BY FIRST_NAME, LAST_NAME
  HAVING COUNT(*) > 5
ORDER BY LAST_NAME ASC NULLS FIRST
   LIMIT 2
  OFFSET 1
     FOR UPDATE
      OF FIRST_NAME, LAST_NAME
```

```
create.select(FIRST_NAME, LAST_NAME, count())
      .from(AUTHOR)
      .join(BOOK).on(Author.ID.equal(Book.AUTHOR_ID))
      .where(LANGUAGE.equal("DE"))
      .and(PUBLISHED.greaterThan(parseDate("2008-01-01")))
      .groupBy(FIRST_NAME, LAST_NAME)
      .having(count().greaterThan(5))
      .orderBy(LAST_NAME.asc().nullsFirst())
      .limit(2)
      .offset(1)
      .forUpdate()
      .of(FIRST_NAME, LAST_NAME)
```

Example taken from www.jooq.org

ScalaQuery

```
case class User(id: Long, email: String, password: String){
    def roles = Roles.where(_.userId === id).list
}

val Users = new Table[User]("USER"){
    def id = column[Long]("ID", O.PrimaryKey)
    def email = column[String]("EMAIL")
    def password = column[String]("PASSWORD")
    def * = id ~ email ~ password <> (User, User.unapply _)

    def roles = List("asdf", "FDSA")
}
```

```
//setup
Users.where(u => u.email === "jane@maxant.ch").delete

//insert
assert(1 == Users.insert(User(1, "jane@maxant.ch", "pswd")))
```

- Compile time checking and type safety
- Abstraction at the SQL level rather than OO
- Supports ORM too, to save you having to write mapping code

What to watch out for

- JSF and JPA work with Java collections
  - Implicit converters are available in Scala
- Breaking changes in Scala
  - Historically there have been some (Collections API)
  - Deprecated stuff will be removed
  - Java should be more harsh like this but…
- Debugging is special
- “problems” view in Eclipse is your good friend

Questions?